AD-A039 874   YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE   F/G 12/2

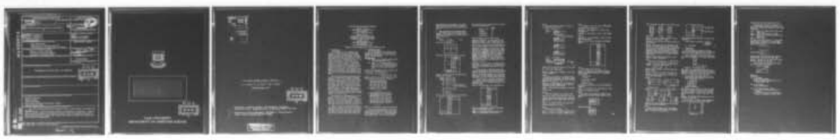
ON AN ARRAY SORTING PROBLEM OF KOSARAJU.(U)

FEB 77 R J LIPTON, R E MILLER, L SNYDER   N00014-75-C-0752

UNCLASSIFIED   RR-95   NL

| OF |

AD
A039874



END

DATE
FILMED
6-77

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 14 RR-95	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER 12
4. TITLE (and Subtitle) 6 On an array sorting problem of Kosaraju.		5. TYPE OF REPORT & PERIOD COVERED 9 Research rept. B. S.
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Richard J. Lipton, Raymond E. Miller, Lawrence Snyder		8. CONTRACT OR GRANT NUMBER(s) 15 N00014-75-C-0752
9. PERFORMING ORGANIZATION NAME AND ADDRESS Yale University, Department of Computer Science, 10 Hillhouse Ave, New Haven, CT 06520		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research, Information Systems Program, Arlington, Virginia 22217		12. REPORT DATE 11 FEB 1977
		13. NUMBER OF PAGES 12 7p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

parallel sorting
array sorting
cellular arrays
bottle neck problem for parallel arrays
local sorting rules

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An array sorting problem is shown to have a linear lower time bound for $n \times n$ ~~$\sqrt{n} \times \sqrt{n}$~~ arrays. This negatively settles a conjecture of Kosaraju. Several other sorting schemes are considered, but all fail to improve performance. The impossibility of better than linear time sorting bounds for arrays with local rules is conjectured.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407 051 - mly

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

On an Array Sorting Problem of Kosaraju

R. J. Lipton,[†] R. E. Miller,[*] and L. Snyder[†]

Research Report #95

[†] Department of Computer Science, Yale University, New Haven, Connecticut 06520. Supported in part by ONR grant N00014-75-C-0752.

[*] Mathematical Sciences Department, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.

ON AN ARRAY SORTING PROBLEM OF KOSARAJU

Richard J. Lipton†
Department of Computer Science
Yale University
New Haven, Connecticut 06520

Raymond E. Miller
Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Lawrence Snyder†
Department of Computer Science
Yale University
New Haven, Connecticut 06520

† Supported in part by ONR contract number N00014-75-C-0752

1. *Introduction*

S. Rao Kosaraju [2] in considering the question of how one can sort on parallel machines was lead to consider parallel machines that are organized into $n \times n$ arrays, i.e., configurations where the (i,j)*th* machine can communicate only with the machines $(i+1,j)$, $(i-1,j)$, $(i,j+1)$, and $(i,j-1)$. The question is, of course, not whether such arrays can sort but rather how fast they can sort. In particular, Kosaraju was lead to a specific simple set of rules for the behavior of the machines and then conjectured that for these rules sorting required at most $O(n)$ time. Our principle result, however, is the failure of this conjecture. Indeed, we will demonstrate that Kosaraju's rules require in worst case at least cn^2 ($c>0$ constant) time.

The fact that Kosaraju's rules fail to sort in $O(n)$ time leads us to consider the general question: are all "local" rules (in a reasonable sense) unable to sort in $O(n)$ time; indeed, do all such rules require at least cn^2 time in worst case. The evidence that we have to support this conjecture is a series of rules similar to Kosaraju's that all have worst case sorting times of cn^2. On the other hand, it is known (Thompson [3]) that there are arrays that can sort in $O(n)$ time. However, these arrays use quite nonlocal rules; hence, they do not contradict our conjecture but instead serve to help delimit the notion of local.

The motivation for studying Kosaraju's conjecture is twofold. First, the question of whether or not a local set of rules can sort in $O(n)$ time is an interesting problem. Second, if there is a set of local rules that can sort in $O(n)$ time, then it may be possible to build hardware that implements directly such an array.

Finally, a note about the organization of the rest of the paper. In section 2 we will present Kosaraju rules and then prove in section 3 that these rules require cn^2 time in worst case. In section 4 we will sketch several other systems of rules with the same worst case behavior.

2. *The Kosaraju Rules*

We will now define the set of rules that Kosaraju had considered. As in Knuth [1], we consider arrays of 0's and 1's rather than arrays of arbitrary numbers.

Kosaraju Rules:

(1) *Bubble:* If a 0 appears directly above a 1, interchange the 0 and 1.

(2) *Snake:* If a 0 appears directly in front of a 1 in a "snake order" on the array, *and* rule (1) does not apply to this 0 or 1, then interchange the 0 and 1.

Letting (i,j) represent the row i, column j cell of the array, the "snake order" is

$$(1,1)(1,2)\ldots(1,n)(2,n)(2,n-1)\ldots(2,1)(3,1)(3,2)\ldots(3,n)$$

ending in (n,n) for n odd and $(n,1)$ for n even. This is shown for $n=4$ and $n=5$ in Figure 1.

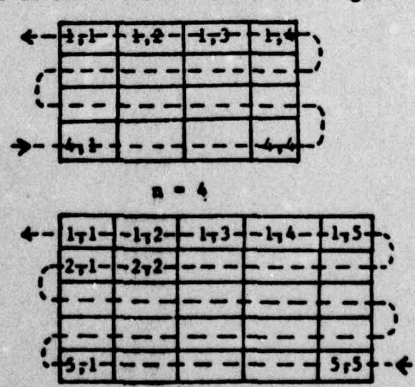


Figure 1: Snake order on boards of $n=4$ and $n=5$.

A step in the sorting process is from the current configuration of 0's and 1's to a new configuration in which *all* interchanges allowed by the rules applied to the current configuration actually occur. Thus, the process does

many simultaneous interchanges. The time required by these rules is then the number of such parallel steps until no rules are applicable.

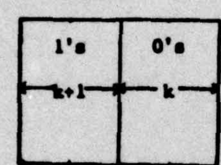
3. *The Lower Bound for the Kosaraju Rules*

Theorem: There exists an n×n binary array requiring cn^2 (c>0) time to sort using the Kosaraju sorting rule.

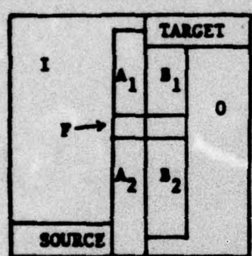
Proof:

Assume: $n = 4i+3$ $i \in N^+$
$k = (n-1)/2$

The array has the form



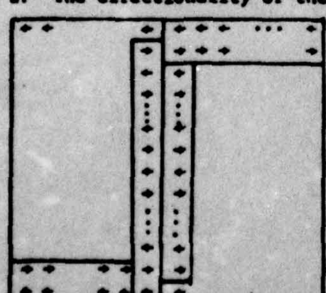
Define the fields of the array as follows:



Fact 1: The sizes and initial states of the fields are as follows:

Source = all 1's = 2×k
target = all 0's = 2×k
A_1 = all 1's = k-1×1
B_1 = all 0's = k-2×1
FF = <2,0> = 1×2 at coordinates (k+1,k+1)(k+1,k+2)
A_2 = all 1's = k×1
B_2 = all 0's = k-1×1
I = all 1's
O = all 0's

Fact 2: The directionality of the fields is



The structure of the computation may be divided into the following pieces:

	Steps
Steps #1 & #2	$\alpha_1=2$
Preamble	α_2
λ-Iterations	α_3
Conclusions	α_4

(the time analysis will be given later).

Overview of behavior: We will claim that the source field emits 1's at a uniform rate, that they move up the A_2 channel at the same rate to the "1/2 way" point, F, *cross over* (smoothly) to the B_1 channel, procede up this channel and are absorbed by the target in a smooth rate. When the source is exhausted, then the two row field above it will be the next source field and that it enters the computation smoothly. When the target fills, the next two row field beneath it becomes the new target and that it does so smoothly. Meanwhile, the I (0) field will remain constant 1's (0's). Finally, during the iteration steps the A_1 field (B_2 field) will be constant 1's (0's).

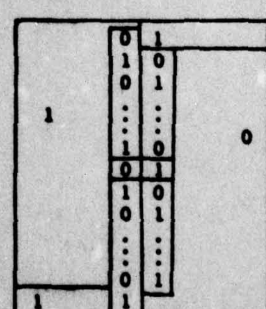
Definition: A field is said to *uniformly emit ones* (ueo) if it delivers a 1 to a fixed cell of the field on alternating steps until it is exhausted.

A field is said to *uniformly absorb ones* (uao) if it accepts 1's from a fixed cell of the field on alternating steps until it is filled.

Remark: The stronger property will be used that ueo's emit 1's only originally in the field and uao's will store the 1's only within the field.

A field is said to *uniformly transmit ones* if it moves 1's from a fixed cell to a fixed cell such that 1's are absorbed (emitted) on alternating steps.

Fact 3: The first step of the computation is:



Note the configuration of TARGET is the basis step in the following lemma.

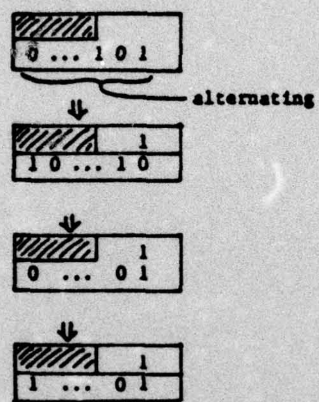
Lemma 1: Target uniformly absorbs ones provided 1's are uniformly emitted from B_1.

Proof:

Using directionality (Fact 2), argue in two steps each by induction.

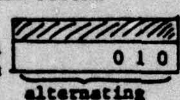
Step 1: fill top

hypoth



Step 2: fill bottom

hypoth
from Step 1
completion



use Floyd [1]

□

Denote the left and right halves of F as F_a and F_b. Call B the column $B_1F_bB_2$.

Lemma 2: During steps 2 through k, B uso from its top position.

Proof:

Since 1's in B have a *0* in them the Kosaraju move takes precedence. By Lemma 1, target can absorb the 1's uniformly so the alternating sequence isn't broken. □

Remark: B actually emits all 1's currently in B at step 1 by this argument, but we are only concerned with the preamble moves.

Lemma 3: During steps 2 through k, A_1 uso provided F_a uso.

Proof:

Since the elements of A_1 alternate they will fill A_1 since the Kosaraju move takes precedence provided they cannot escape. By directionality (Fact 2) and Lemma 2 this cannot happen. □

Lemma 4: In steps 2 through k, A_2 emits ones uniformly provided F_a is a uso.

Proof:

K rule takes precedence. □

Fact 4: The source is constant in steps 2 and 3.

Lemma 5: In steps 4 through k, SOURCE is uso.

Proof:

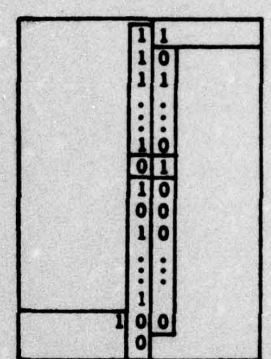
By Lemma 4, A_2 will be a uso. SOURCE emits from upper right hand corner. □

Corollary 1: (of Lemmas 3 & 4) F_a is a uto, through step k.

Corollary 2: (of Lemma 2) B_2 is a uso, through step k.

Corollary 3: (of all lemmata) I and O are constant through step k.

Lemma 6: After k steps (preamble) the configuration is:



Proof:

Apply lemmas and corollaries. □

Remark: The computation is "stable" hereafter until the interaction completes. The behavior is as follows:

Source is uso
A_2 is uto
F is uto (by k moves)
B_1 is a uto
target is a uso.

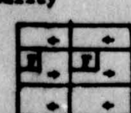
We now argue the behavior of F and then describe iteration.

Lemma 7: As long as A_2 is a uso and B_1 is a uso, then F is a uto until k moves.

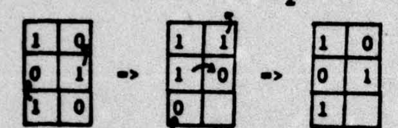
Proof:

By Lemma 6 and hypotheses of lemma, we may consider only the local behavior which is initially:

with directionality



The transitions are (given A_2 is ueo and B_1 uao)



hypoth
step (odd) (even) (odd)

Fact 5: At step k, target contains $\ell+1$ ones and source has emitted $\ell-1$ ones.

Definition: An iteration is completed when target is filled.

Remark: The results thus far demonstrate that the 1's will stream uniformly into target until it fills. We are seeking to show that once target is filled, a new one is started without breaking cadence; and analogously for sources. Before the actual argument we first discuss the timing in anticipation of the complexity arguments later.

Lemma 8: Target requires 4k-2 steps to fill from its initial configuration.

Proof:

The initial configuration is $\boxed{1\qquad\qquad}$

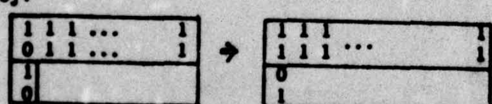
A total of 2k 1's are required of which one is present. It takes 2 steps to each 1.

$$2(2k-1) = 4k-2 \qquad \square$$

Remark: Note, initially 1 step is required to get in "phase."

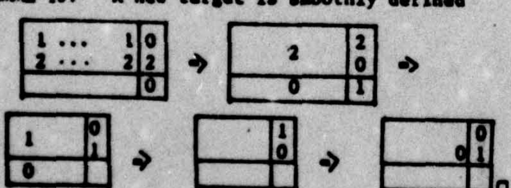
Lemma 9: If B_1 is uto, then on the step that target is filled, a new target is defined and in initial configuration.

Proof:



Step before fill initial fill $\square$

Lemma 10: "A new target is smoothly defined"

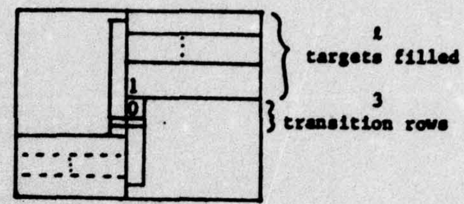


$\square$

Finally, the final configuration is not reached until cn^2 steps (c>0 constant).

Proof:

The conclusion begins after ℓ iterations, where the first iteration contains the preamble steps. The state at conclusion is



The three transition rows will be filled in the conclusion.

$$\text{Total steps to conclusion} = \ell(4k-2)+1 \quad \text{by Lemma 7}$$
$$= 8\ell^2+2\ell+1$$

$$n = 16\ell^2+24\ell+9 \qquad \square$$

4. *Other Rules*

In this section we present two further rules and show (or rather sketch) that they both require cn^2 time in the worst case to sort.

In the first set of rules we consider that the array is organized as a cylinder (see Figure 2).

Figure 2: The cylinder array.

One column is distinguished and is called the *date line*. The rules are then bubble if you can; otherwise, shift to the left if you can (just as before). In addition, we add a new rule:

> A 1 can wrap around (see Figure 3), i.e., cross the data line going to the left provided the position α is a 0.

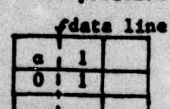
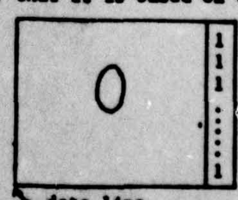


Figure 3: New rule.

The rationale behind this rule is that it seemed to allow the array to break up large blocks of 1's that are packed to the left as in the Kosaraju counterexample. However, this is not the case!

Theorem: The cylinder scheme requires cn^2 (c>0 constant) in worst case.

We will omit the proof of this theorem and remark only that it is based on the example



and shows that the time required is n^2-n+1.

Our third set of rules is one based on a set of interchange rules due to Floyd [1]. We assume that n is odd and we operate on the array as follows:

Step 1: Apply left (right) Floyd rules to odd (even) rows.
Step 2: Apply up Floyd rules to columns.
Step 3: Apply right (left) Floyd rules to odd (even) rows.
Step 4: Repeat step 2.

The right (left) Floyd rule is: a 1 can interchange with a 0 that is to its left (right). The up version is defined in a similar manner. Thus, the left Floyd on

0 1 0 1 0 1 1 1

is

1 0 1 0 1 0 1 1

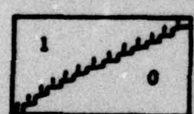
and the right Floyd is

0 0 1 0 1 1 1 1.

Although these rules are less local than our previous ones, they suffer the same fate:

Theorem: The above set of rules sorts in time cn^2 ($c>0$ constant) in worst case.

Again we omit the proof and just remark that it is based on the example



References

[1] D. E. Knuth.
Sorting and Searching.
The Art of Computer Programming, Vol. 3, Addison Wesky, 1975.

[2] S. R. Kosaraju.
Private communication.

[3] C. D. Thompson and H. T. Kung.
Sorting on a Mesh-Connected Parallel Computer.
Proc. of the 8th Annual ACM Symposium on Theory of Computing, pp 58-64, 1976.